

Examen de Prácticas de Programación (Ingeniería Informática)

Junio 2006

Parte I. Cuestiones (3 puntos=50% nota del examen)

- 1) Se desea crear un conjunto de clases para representar en un programa las personas pertenecientes a la Universidad. Existen tres tipos de universitarios: alumnos, profesores, y personal de administración y servicios (PAS). Los dos últimos son empleados de la Universidad (los alumnos no).

Se pide hacer el diseño arquitectónico de una jerarquía de clases que representen a estas personas, teniendo en cuenta los siguientes datos (todos de tipo texto) y operaciones:

- Todos los universitarios tienen nombre y dirección, y operación para mostrar datos
- Los alumnos tienen centro y operación para matricularse en un centro
- Los profesores tienen departamento y operación para adscribirse a él
- Los PAS tienen unidad administrativa y operación para adscribirse a ella
- Los empleados tienen número de registro y operación para obtenerlo

Se pide también el diseño detallado de los constructores de cada clase, a los que se pasarán como parámetros los valores iniciales de cada atributo.

- 2) Decimos que una pila p es sombrero de otra pila q si todos los elementos de p están en q en el mismo orden y en las posiciones más próximas a la cima. La pila nula se considera sombrero de cualquier otra pila. Por ejemplo, si $A:=[7,9,4,2]$, $B:=[7,9,4,2,5,3,6]$ y $C:=[7,9,2,4,5,3,6]$, entonces A es sombrero de la pila B , pero no de la pila C .

Se supone que el tipo genérico `elemento` dispone de una operación de comparación: `x.equals(y)` que aplicado a dos elementos x e y retorna cierto si y sólo si x e y son iguales. Se pide:

- Especificar y diseñar un método estático recursivo `esSombrero(p,q)` que determine si la pila p es sombrero de la pila q .
- Diseñar el método anterior mediante un bucle.

- 3) Se dispone de la siguiente clase ya realizada, que representa una red de comunicación:

```
/** Clase que contiene una operación para recibir  
 * mensajes de texto por un canal de comunicación */  
public class CanalComunicacion  
{  
    /**  
     * Recibir un mensaje de texto del puerto y emisor indicados  
     */  
    public String recibe(int puerto, int emisor) throws NoDisponible  
    {...}  
}
```

Se pide escribir un método para esa clase, con los mismos parámetros y valor de retorno que `recibe()` y que llame a éste un máximo de 5 veces (usando en la llamada los parámetros obtenidos). Si alguna de las llamadas tiene éxito (porque no se lanza `NoDisponible`), finalizar el método retornando el valor retornado a su vez por `recibe()`. Después del 5 intento sin éxito, lanzar la excepción `Imposible`. Ambas excepciones están declaradas en clases aparte en la forma:

```
public class NoDisponible extends Exception {}  
public class Imposible extends Exception {}
```

4) Escribir en Java la implementación del método inserta() de la clase siguiente:

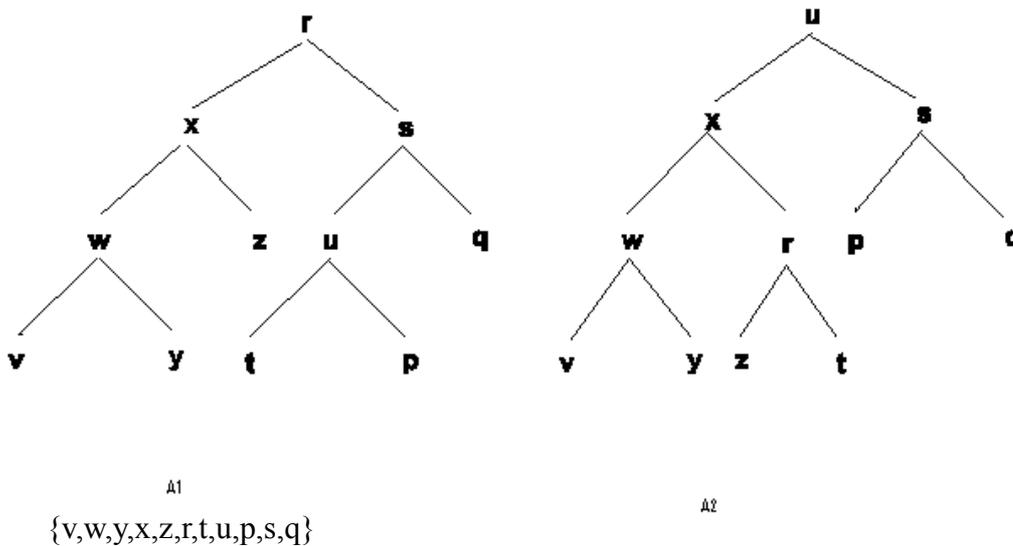
```

/**
 * Clase que representa una lista de nombres
 */
public class ListaNombres
{
    // atributo, que contiene la lista
    private LinkedList<String> lista=new LinkedList<String>();

    /**
     * Inserta sin repeticiones los strings del array nombres
     * en la lista, añadiéndolos uno por uno al final. Aquellos que
     * ya están en la lista no se añaden.
     */
    public void inserta(String[] nombres)
    { ... }
}

```

5) Un árbol binario (como el A1 o el A2 de la figura) puede recorrerse de muchas formas, en particular en inorden, es decir, primero se lista el hijo izquierdo en inorden, luego la raíz y luego el hijo derecho en inorden. Esta definición recursiva se completa definiendo el inorden del árbol vacío como la lista vacía. En la figura se muestra a modo de ejemplo el recorrido en inorden de A1.



Responder a la siguiente pregunta: ¿puede recuperarse un árbol binario conociendo la lista de sus nodos en inorden?. Si la respuesta es negativa exhibir un contraejemplo. Si es afirmativa argumentar la razón.

Examen de Prácticas de Programación (Ingeniería Informática)

Junio 2006

Parte II. Problema (3 puntos=50% nota del examen)

Se desea hacer parte del programa de gestión del inventario de los artículos que se venden en una tienda. Para ello, se dispone de la siguiente clase ya realizada, cuya interfaz es:

```
/**
 * Clase que representa un artículo que se vende en una tienda
 */
public class Artículo
{
    /**
     * Constructor al que se le pasan los datos del artículo
     */
    public Artículo(String nombre, int cantidad, double precioUnidad) {...}

    /**
     * Retorna el nombre
     */
    public String nombre() {...}

    /**
     * Retorna el número de unidades disponibles
     */
    public int cantidad() {...}

    /**
     * Retorna el precio de una unidad
     */
    public double precioUnidad() {...}

    /**
     * Vende num artículos (restándolos de la cantidad disponible), si los
     * hay. En caso contrario, no modifica la cantidad y
     * lanza NoDisponible.
     */
    public void vende(int num) throws NoDisponible {...}
}
```

Lo que se pide es hacer una clase que almacene la lista de artículos y que responda a la siguiente interfaz

```
/**
 * Clase que contiene una lista con los artículos en venta en una tienda
 */
public class Tienda
{
    /**
     * Constructor al que se le pasa el nombre de la tienda
     * y que deja la lista vacía
     */
    public Tienda(String nombre) {...}

    /**
     * Añade un artículo a la lista
     */
    public void inserta (Articulo a) {...}

    /**
     * Busca y retorna el artículo cuyo nombre se indica
     * Lanza NoExiste si no se encuentra
     */
    public Articulo busca (String nombre) throws NoExiste {...}
}
```

```

/**
 * Vende num unidades del artículo del nombre indicado. Para ello busca
 * el artículo en la lista y usa su método vende. Si no encuentra
 * el artículo lanza NoExiste. Si la cantidad queda a cero o si se ha
 * lanzado NoDisponible, elimina el artículo de la lista. Retorna el
 * valor total de las unidades vendidas.
 */
public double vende (String nombre, int num) throws NoExiste {...}

/**
 * Escribe en el fichero de texto cuyo nombre se indica
 * aquellos artículos de la lista de los que queden menos
 * de min unidades. Se escriben uno por línea, cada uno
 * con sus tres datos. En la primera línea se escribirá el
 * nombre de la tienda
 */
public void escribePedido(String nombreFichero, int min) {...}
}

```

La clase tendrá como atributos privados una lista de artículos (usar por ejemplo la clase LinkedList), y el nombre de la tienda (texto).

Las excepciones están declaradas en clases aparte en la forma:

```

public class NoDisponible extends Exception {}
public class NoExiste extends Exception {}

```

Nota: Se valorará cada parte de la clase Tienda según su dificultad.